Lab program 1

Aim:

Writing a program to check whether the given number is Armstrong or not.

Source code:

```
n = int(input("Enter a number: "))
temp = n
tot = 0
num_digits = len(str(n))
while n != 0:
    r = n % 10
    tot += r ** num_digits
    n //= 10
if temp == tot:
    print(f"{temp} is an Armstrong number")
else:
    print(f"{temp} is not an Armstrong number")
```

output:

```
Enter a number: 9474
9474 is an Armstrong number
```

Lab program 2

Aim:

Writing a program to check whether the given number is Perfect or not.

Source code:

```
def is_perfect(number):
    sum_of_divisors = sum(i for i in range(1, number) if number % i == 0)
    return sum_of_divisors == number

num = int(input("Enter a number: "))

if is_perfect(num):
    print(f"{num} is a perfect number")
else:
    print(f"{num} is not a perfect number")
```

output: Enter a number: 10

10 is not a perfect number

Lab program 3

Aim:

Writing a program to find factorial of given number using recursive function.

Source code:

```
def recursive_factorial(n):
  if n == 1:
      return n
  else:
      return n * recursive_factorial(n-1)
num = 6
if num < 0:
      print("Invalid input! Please enter a positive number.")
elif num == 0:
      print("Factorial of number 0 is 1")
else:
      print("Factorial of number", num, "=", recursive_factorial(num))
```

output:

```
Factorial of number 6 = 720
```

Lab program 4

Aim:

Writing a program to implement inheritance and polymorphism.

Source code:

```
class Operations:
    def __init__(self):
        self.a = 10
        self.b = 20
        self.c = "vinam"
        self.d = "bratha"

    def execute(self):
        pass

class Addition(Operations):
    def execute(self):
        print("Addition:", self.a + self.b)

class Concat(Operations):
    def execute(self):
        print("Concatenation:", self.c + self.d)



add_obj = Addition()
concat_obj = Concat()

add_obj.execute()
concat_obj.execute()
```

output:

Addition: 30

Concatenation: vinambratha

Lab program 5

Aim:

Demonstrating a python code to print try, except and finally block statements

Source code:

```
try:
    print("Try Block: Attempting to divide...")
    result = 10 / 0
except ZeroDivisionError:
    print("Except Block: Cannot divide by zero!")
finally:
    print("Finally Block: This will always execute.")
```

output:

Try Block: Attempting to divide...

Except Block: Cannot divide by zero!

Finally Block: This will always execute.

Lab program 6

Aim:

Writing a program to demonstrate String handling functions.

Source code:

```
def menu():
    while True:
        print("\nString Handling Functions")
        print("1.String Length")
        print("2.Slicing String")
        print("3.UPPER CASE")
        print("4.lower case")
        print("5.String Concatenation")
        print("0.Exit")
        choice=input("Enter your choice: ")
        if choice=='1':
            print("\n")
            string=input("Enter a String: ")
            print("String Length: ",len(string))
        elif choice=='2':
            print("\n")
            string=input("Enter a String: ")
            start=int(input("Enter starting index: "))
            end=int(input("Enter ending index: "))
            print("Slicing String: ",string[(start):(end)])
        elif choice=='3':
            print("\n")
            string=input("Enter a String: ")
            print("UPPER CASE: ", string.upper())
        elif choice=='4':
            print("\n")
            string=input("Enter a String: ")
            print("lower case: ",string.lower())
        elif choice=='5':
            print("\n")
            string1=input("Enter First String: ")
            string2=input("Enter Second String: ")
            print("String Concatenation: ",string1+string2)
        elif choice=='0':
```

```
            print("Exiting...")
            break
        else:
            print("Invalid! try agin...")
menu()
```

output:

output for choice: 1

Enter a String: Lightning McQueen

String Length: 17

output for choice: 2

Enter a String: Lightning McQueen

Enter starting index: 0

Enter ending index: 10

Slicing String: Lightning

output for choice: 3

Enter a String: Lightning McQueen

UPPER CASE: LIGHTNING MCQUEEN

output for choice: 4

Enter a String: LIGHTNING MCQUEEN

lower case: lightning mcqueen

output for choice: 5

Enter First String: Lightning McQueen

Enter Second String: Ka-chow!

String Concatenation: Lightning McQueen Ka-chow!

output for choice: 0

Exiting...

Lab program 7

Aim:

Writing a program to input n numbers from the user. Store these numbers in a tuple. Print the maximum and minimum number from this tuple.

Source code:

```
n = int(input("Enter the number of elements: "))

num_list = [int(input(f"Enter number {i+1}: ")) for i in range(n)]

numbers = tuple(num_list)

print("\nTuple of numbers:", numbers)
print("Maximum number:", max(numbers))
print("Minimum number:", min(numbers))
```

output:

Enter the number of elements: 3

Enter number 1: 45

Enter number 2: 79

Enter number 3: 2

Tuple of numbers: (45, 79, 2)

Maximum number: 79

Minimum number: 2

Lab program 8

Aim:

Write a program to enter names of employees and their salaries as input and store them in a dictionary

Source code:

```
employees={}
n = int(input("Enter the number of employees: "))
for i in range(n):
    name=input(f"Enter the name of employee {i+1}: ")
    salary=int(input(f"Enter the salary of {name}: "))
    employees[name]=salary
print("\nPrinting Dictionary: ")
print(employees)
print("\nPrinting Dictionary in a structured format: ")
print("\nEmployee Salary Data:\n")
print("Emp Name\tEmp Salary")
print("..................................")
for name,salary in employees.items():
    print(f"{name}\t\t{salary}")
```

output:

Enter the number of employees: 3

Enter the name of employee 1: sethu

Enter the salary of sethu: 35999

Enter the name of employee 2: vardhan

Enter the salary of vardhan: 99999

Enter the name of employee 3: ram

Enter the salary of ram: 150000

Printing Dictionary:

{'sethu': 35999, 'vardhan': 99999, 'ram': 150000}

Printing Dictionary in a structured format:

Employee Salary Data:

Emp Name	Emp Salary
sethu	35999
vardhan	99999
ram	150000

Lab program 9

Aim:

Writing a program to implement statistical operations on arrays using NumPy

Source code:

```
import numpy as np
arr1 = np.array([2,10,6,8])
arr2 = np.array([2,2,3,2])
newarr = np.add(arr1, arr2)
print(newarr)
newarr = np.subtract(arr1, arr2)
print(newarr)
newarr = np.multiply(arr1, arr2)
print(newarr)
newarr = np.divide(arr1, arr2)
print(newarr)
newarr = np.power(arr1, arr2)
print(newarr)
print(min(arr1))
print(max(arr1))
```

output:

[4 12 9 10]

[0 8 3 6]

[4 20 18 16]

[1. 5. 2. 4.]

[4 100 216 64]

2

10

Lab program 10

Aim:

Writing a program to import and export CSV file to Data Frame.

Source code:

```
import pandas as pd

data = {
  "character": ["Lightning McQueen", "Mater", "Doc Hudson"],
    "car_model": ["Stock Car", "Tow Truck", "Hudson Hornet"],
    "role": ["Racer", "Tow Truck Driver", "Retired Racer"]
}

df = pd.DataFrame(data)

csv_filename = "cars.csv"
df.to_csv(csv_filename, index=False)

print(f"Data exported to {csv_filename}")

df_imported = pd.read_csv(csv_filename)

print("Imported DataFrame:")
print(df_imported)
```

output:

Data exported to cars.csv

Imported DataFrame:

	character	car_model	role
0	Lightning McQueen	Stock Car	Racer
1	Mater	Tow Truck	Tow Truck Driver
2	Doc Hudson	Hudson Hornet	Retired Racer

Lab program 11

Aim:

Create the Data Frame Sales containing year wise sales and perform basic operation on it.

Source code:

```
import pandas as pd

data = {
    'year': [2015, 2016, 2017, 2018, 2019],
    'sales': [45000, 52000, 61000, 58000, 67000]
}

df = pd.DataFrame(data)

print("Sales DataFrame:\n", df)

print("\nBasic Statistics:\n", df.describe())

total_sales = df['sales'].sum()
print("Total Sales:", total_sales)

average_sales = df['sales'].mean()
print("Average Sales:", average_sales)

max_sales = df['sales'].max()
print("Maximum Sales:", max_sales)

min_sales = df['sales'].min()
print("Minimum Sales:", min_sales)

df['sales_after_increase'] = df['sales'] * 1.1
print("\nDataFrame after 10% increase in sales:\n", df)
```

output:

Sales DataFrame:

	year	sales
0	2015	45000
1	2016	52000
2	2017	61000
3	2018	58000
4	2019	67000

Basic Statistics:

	year	sales
count	5	5
mean	2017	56600
std	1.58	8443.93
min	2015	45000
25%	2016	52000
50%	2017	58000
75%	2018	61000
max	2019	67000

Total Sales: 283000

Average Sales: 56600

Maximum Sales: 67000

Minimum Sales: 45000

DataFrame after 10% increase in sales:

	year	sales	sales_after_increase
0	2015	45000	49500.0
1	2016	52000	57200.0
2	2017	61000	67100.0
3	2018	58000	63800.0
4	2019	67000	73700.0

Lab program 12

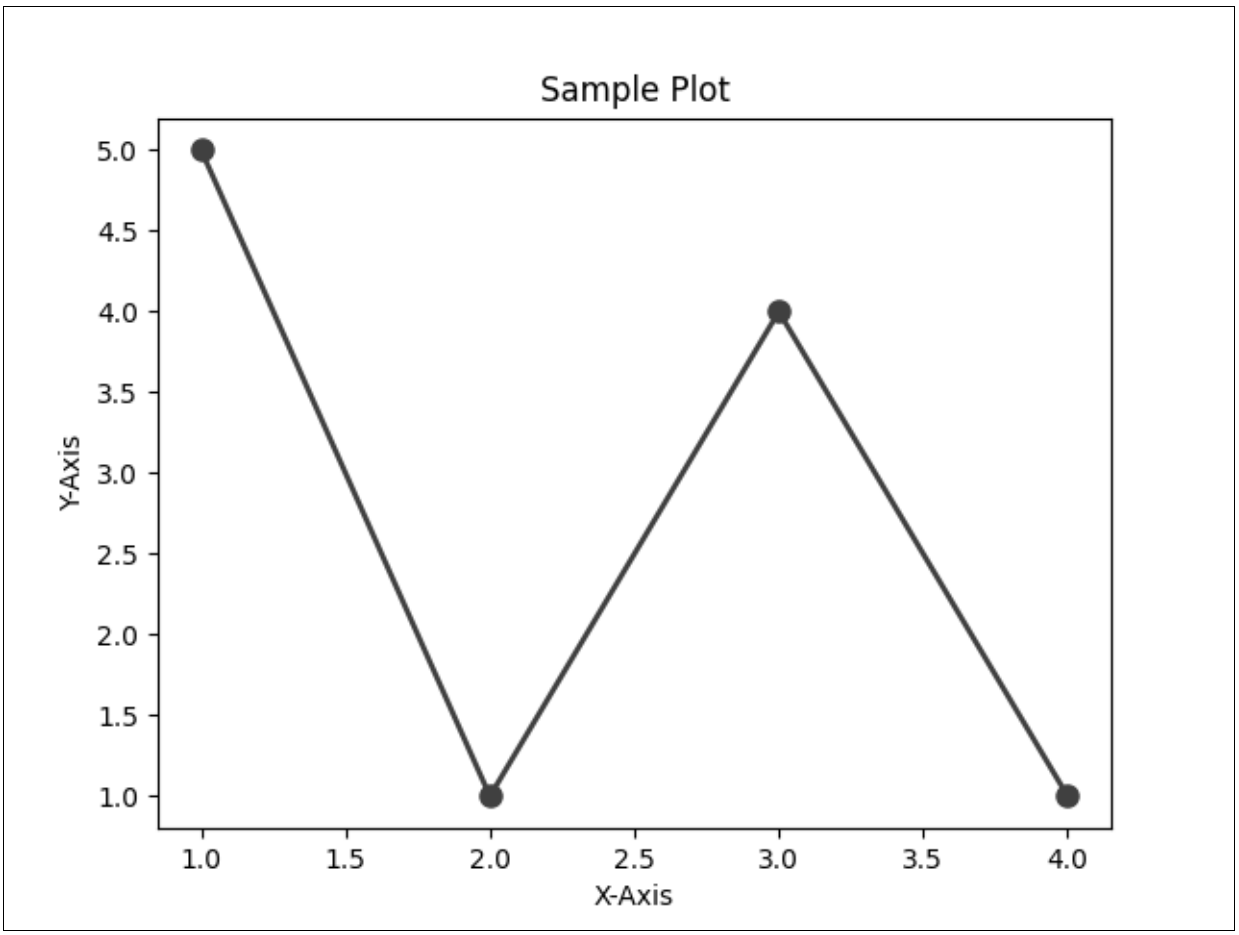
Aim:

Visualize the plots using matplotlib.

Source code:

```
import matplotlib.pyplot as plt



x = [1, 2, 3, 4]
y = [5, 1, 4, 1]

plt.plot(x, y, marker='o', linestyle='-', color='black', linewidth=2, markersize=8)

plt.xlabel('X-Axis')
plt.ylabel('Y-Axis')
plt.title('Sample Plot')
plt.show()
```

output:


Sample Plot
Y-Axis
X-Axis
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
1.0
1.5
2.0
2.5
3.0
3.5
4.0

Lab program 13

Aim: Write a program to connect with MySQL data base and perform CRUD (Create, Read, Update and Delete) operations

Source code:

```
import mysql.connector



conn = mysql.connector.connect(
    host="localhost",
    user="your username",
    password="your mysql password",
    database="your database name"
)

cursor = conn.cursor()



cursor.execute("""
    CREATE TABLE IF NOT EXISTS shows (
        id INT AUTO_INCREMENT PRIMARY KEY,
        show_name VARCHAR(255) NOT NULL,
        genre VARCHAR(100) NOT NULL,
        episodes INT NOT NULL
    )
""")
print("Table 'shows' created successfully!")



shows_data = [
    ("Man vs Wild", "Adventure", 120),
    ("Food Factory", "Food", 80),
    ("Snakes in the City", "Wildlife", 50),
    ("Deadliest Catch", "Fishing", 200),
    ("MythBusters", "Science", 300)
]

cursor.executemany("INSERT INTO shows (show_name, genre, episodes) VALUES (%s, %s,
%s)", shows_data)
conn.commit()
```

```
print("Inserted Discovery Channel shows successfully!")


cursor.execute("SELECT * FROM shows")
print("\nDiscovery Channel Shows:")
for row in cursor.fetchall():
    print(row)

cursor.execute("UPDATE shows SET episodes = 130 WHERE show_name = 'Man vs Wild'")
conn.commit()
print("\nUpdated 'Man vs Wild' episode count.")


cursor.execute("DELETE FROM shows WHERE show_name = 'Snakes in the City'")
conn.commit()
print("\nDeleted 'Snakes in the City' from database.")


cursor.execute("SELECT * FROM shows")
print("\nFinal Shows List:")
for row in cursor.fetchall():
    print(row)


cursor.close()
conn.close()
print("\nDatabase connection closed.")
```

output:

```
Table 'shows' created successfully!
Inserted Discovery Channel shows successfully!

Discovery Channel Shows:
(1, 'Man vs Wild', 'Adventure', 120)
(2, 'Food Factory', 'Food', 80)
(3, 'Snakes in the City', 'Wildlife', 50)
(4, 'Deadliest Catch', 'Fishing', 200)
(5, 'MythBusters', 'Science', 300)

Updated 'Man vs Wild' episode count.

Deleted 'Snakes in the City' from database.

Final Shows List:
(1, 'Man vs Wild', 'Adventure', 130)
(2, 'Food Factory', 'Food', 80)
(4, 'Deadliest Catch', 'Fishing', 200)
(5, 'MythBusters', 'Science', 300)

Database connection closed.
```